G-MPSoC: Generic Massively Parallel Architecture on FPGA

Hana Krichene, Mouna Baklouti, Mohamed Abid, Philippe Marquet, Jean-Luc Dekeyser

G-MPSoC: Generic Massively Parallel Architecture on FPGA

HANA KRICHENE
University of Lille 1
INRIA Lille Nord Europe
Lille - France
ENIS School
CES Laboratory
Sfax - Tunisia
hana.krichene@inria.fr

MOUNA BAKLOUTI
MOHAMED ABID
ENIS School
CES Laboratory
Sfax - Tunisia
mouna.baklouti@enis.rnu.tn
mohamed.abid@enis.rnu.tn

PHILIPPE MARQUET
JEAN-LUC DEKEYSER
University of Lille 1
INRIA Lille Nord Europe
Lille - France
Philippe.Marquet@univ-lille1.fr
jean-luc.dekeyser@univ-lille1.fr

Abstract: Nowadays, recent intensive signal processing applications are evolving and are characterized by the diversity of algorithms (filtering, correlation, etc.) and their numerous parameters. Having a flexible and programmable system that adapts to changing and various characteristics of these applications reduces the design cost. In this context, we propose in this paper Generic Massively Parallel architecture (G-MPSoC). G-MPSoC is a System-on-Chip based on a grid of clusters of Hardware and Software Computation Elements with different size, performance, and complexity. It is composed of parametric IP-reused modules: processor, controller, accelerator, memory, interconnection network, etc. to build different architecture configurations. The generic structure of G-MPSoC facilitates its adaptation to the intensive signal processing applications requirements. This paper presents G-MPSoC architecture and details its different components. The FPGA-based implementation and the experimental results validate the architectural model choice and show the effectiveness of this design.

Key–Words: SoC, FPGA, MPP, Generic architecture, parallelism, IP-reused

1 Introduction

The intensive signal processing applications are increasingly oriented to specialized hardware accelerators, which allow rapid treatment for specific tasks. However, these applications are characterized by repetitive tasks performing multiple data, which require massive parallelism for their efficient execution. To achieve high performance required by these applications, many massively parallel System-on-Chips are proposed [1, 7, 5, 2]. Despite their effectiveness, these solutions are still dedicated to specific applications and it is generally difficult to make future changes to adapt new applications. To address this problem, this paper proposes a novel Generic Massively Parallel System-on-Chip, named G-MPSoC. This system, based on modular structure, allows building different configurations to cover a wide range of intensive signal processing applications. The architectural model of G-MPSoC can integrate software homogeneous computation units or hardware(accelerator)/software(processor) heterogeneous computation units. This generic feature allows the best partition of the specific tasks on the computational resources. To implement this system, the FPGA platform is targeted to exploit its reconfigurable structure, which facilitates the test of different G-MPSoC configurations with rapid re-design or circuit layout modifications. In this work, Xilinx Virtex6 ML605 board is used to implement the G-MPSoC architecture and to evaluate the experimental results.

The remainder of the paper is structured as follows: Section 2 discusses some related works; section 3 describes the proposed architecture and its execution model; the implementation methodology on FPGA board is detailed in Section 4; then, the experimental results are discussed in Section 5; and finally, Section 6 concludes the paper and proposes some perspectives.

2 Related work

Nowadays, the digital embedded systems migrate toward the massively parallel on-Chip design, due to its provided high performance. Among these systems, we note the General-Purpose Processing on Graphics Processing Units (GP-GPU) [2, 24], which is a massively parallel System-on-Chip based on a hybrid model between vector processing and hardware threading. It allows high performance by hiding the memory latency [1]. But, it loses efficiency with the large branch divergence between executing threads [3] when data dependency is demanded in intensive signal (image, sound, motion...) processing applications. In this gap, Platform 2012 (P2012) [1] is positioned

with highly coupled building blocks based on cluster of extensible processors varying from 1 to 16 and shared the same memory. Despite its high scalability, this architecture still specialized for limited range of applications such as multi-modal sensor fusion and image understanding. To provide more flexibility, a heterogeneous extension to the P2012 platform is proposed with He-P2012 [6] architecture. In He-P2012, the clusters of PEs used in P2012 are tightly coupled with hardware accelerators. All of them share the same data memory. With this architecture, a programming model is proposed, allowing the dispatch of Hardware and Software tasks with the same delay. Enlarge the initial software platform by additional hardware blocks can increase the design complexity with multiple communication interfaces. Others architectures have adopted the same strategy of massively parallel shared memory architecture, such as STM STHORM [1], Kalray MPPA [7], Plurality HAL [8], the NVIDIA Fermi GPU [9], etc. Although their provided high performance, sharing the same memory unit can limit the system bandwidth and cause some data access congestion, which limit the system scalability. The autonomous control structure for massively parallel System-on-Chip is proposed with the MPPA [4] architecture and Event-Driven Massively Parallel Fine-Grained Processor Array [25], executing in MIMD fashion. They provide more flexibility with asynchronous execution than the previous centralized and synchronous architectures. But, the use of a completely decentralized processing structure makes the control task of data transfer between independent computation units difficult to achieve.

To overcome the limits of the existing proposed massively parallel architectures on-chip, we define a Generic Massively Parallel SoC (G-MPSoC), allowing the execution of wide range of intensive signal processing applications. To meet the high performance requirements of these applications, this generic architecture can have different configurations going from simple homogeneous structure to clustered heterogeneous structure communicated through regular Network-on-Chip (NoC) and controlled by hierarchical master-slave control structure [14]. This design is implemented into FPGA platform, which allows rapid reconfiguration of the architecture according to the designer needs. This reconfigurability based on programmable logic elements facilitates the system scalability.

In the next section, we detail the major components of G-MPSoC.

3 Generic Massively Parallel Architecture based on Synchronous Communication Asynchronous Computation

3.1 G-MPSoC architecture overview

The G-MPSoC architecture is composed of a Master Controller Unit (MCU) connected to its sequential instructions memory, called MCU-memory, and a grid of Slave Control Units (SCUs). Each SCU is connected to a cluster of 16 Computation Elements (CEs), known collectively as Node. The CE can be a Software Processing Element (PE) or a Hardware specialized Accelerator-IP. Each CE is connected to its local instructions and data memories, called *Mi* memory. The parameter *i* is relative to the CE number. The MCU and SCUs grid are connected through a bus with single hierarchical level and the SCUs are connected together through neighbourhood interconnection network. Fig. 1 shows the hardware implementation of the G-MPSoC. Bellow, we present the design of the G-MPSoC components.

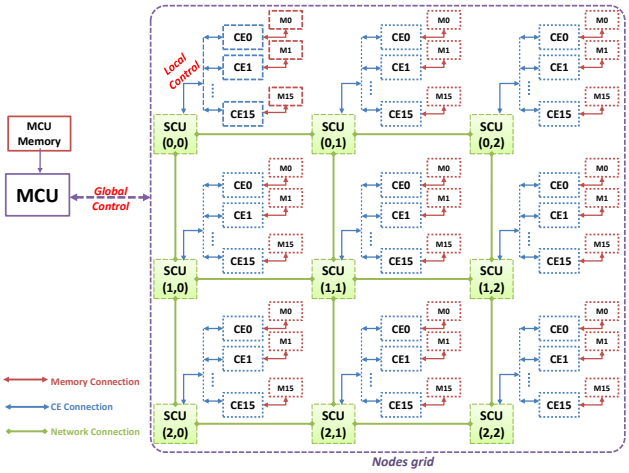

Figure 1: G-MPSoC architecture

3.1.1 MCU

The MCU is the first execution and control level in G-MPSoC. It is a simple processor, which fetches and decodes program instructions, executes sequential instructions and broadcasts mask activity and parallel control instructions to SCUs. In a parallel execution, MCU remains in the idle state and controls the end signal to resume the main program execution. In G-MPSoC, the MCU is based on modified FC16 [10], which is a stack processor with short 16-bits instructions. It is open source and fully implemented in VHDL language, which allows rapid prototyping in

FPGA with instructions set simple to expand. Some specific instructions are added to the FC16 instructions set, mainly:

- Mask instructions
  To activate the involved nodes in the parallel execution, the MCU executes mask instructions, as presented in table 1, and broadcasts the mask map and mask code to all SCUs. The mask is coded in 32 bits, to address an array of (16x16) nodes.

Table 1: Mask instructions

instruction	OpCode	Mask Code	Definition
selbf	0x0080	(000)	Activate SCUs selected by the mask.
selbfand	0x0081	(001)	Activate SCUs in the intersection of the current mask and previous one.
selbfor	0x0082	(010)	Activate SCUs in the union of the current mask and previous one.
selbfxor	0x0083	(011)	Activate SCUs in the union of the current mask and previous one except the intersection part.

- Broadcast instructions
  Coded in 32 bits, the broadcast instructions identify which area executes parallel control instructions (table 2). Once the mask is mapped into the SCUs grid and when the broadcast instruction is executed, the MCU fetches the parallel control instruction, and then broadcasts it to all selected SCUs followed by the broadcast code.

Table 2: Mask instructions

instruction	OpCode	Mask Code	Definition
brdbf	0x0084	(100)	Broadcast parallel control instructions to active SCUs.
brdbfb	0x0085	(101)	Broadcast parallel control instructions to inactive SCUs.
brdall	0x0086	(111)	Broadcast parallel control instructions to all the SCUs.

- Wait_end instruction
  When executing this instruction, the MCU remains in idle state until the end of the parallel execution. It is a synchronising instruction, which depends on the end-execution signal value.

The MCU is connected to the SCUs grid through a bidirectional bus to communicate the parallel control instructions from MCU to SCUs and the computation results from CEs to MCU via SCUs. The end signal connects all SCUs to inform the MCU of the end of the parallel execution.

3.1.2 SCUs array

The second execution and control level in G-MPSoC is presented by the SCUs array. Each SCU controls: the local node activity, the autonomous parallel execution in the CEs, the end execution and the synchronous communication in the interconnection network. As shown in fig. 2, it is composed of four modules, where each one of them independently performs a specific control function, as detailed in [14].

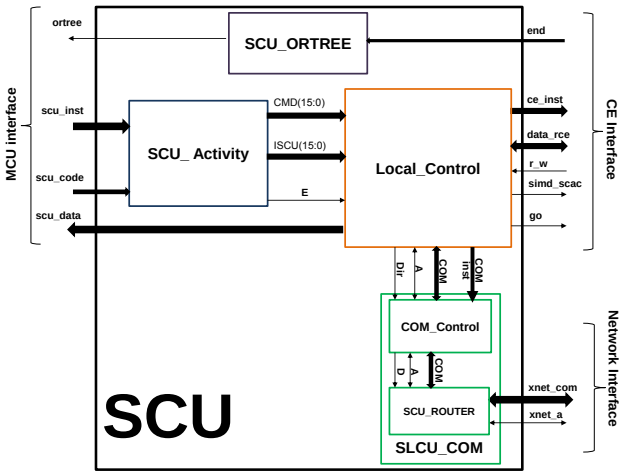

Figure 2: SCU design

- SCU_Activity module
  This module receives the mask activity and the parallel control instruction from the MCU. Then, according to the mask code and broadcast code, it sets the activity-flag and transfers the parallel control instruction to the Local_Control module, respectively. We notice that the use of SCU_Activity module in G-MPSoC architecture allows the sub-netting of the SCUs grid, which optimizes the data flow transfer and increases the parallel broadcast domains.

- Local_Control module
  After sub-netting the network, the parallel control instructions are broadcast to a Local-Control

module. This module prepares the communication phase and controls the CEs execution. The main sub-module in Local-Control is the Instruction Decoder, which decodes the instructions received from the SCU_Activity module. These instructions are coded in 32 bits, as detailed in [14]: the first 16 bits represent the control micro-instruction (CMD) and the last 16 bits represent: the address of the parallel instructions block, the single parallel SIMD instruction (P_INST) or the value of the communicated data.

- SCU_COM module
  The SCU components in G-MPSoC are connected in two-dimensional neighbourhood interconnection network via the SCU-COM module. It allows the SCU to communicate with its 8 neighbours using only 6 connections. The SCU-COM is composed of COM_Control and SCU_router sub-modules. The COM_Control sub-module manages the data transfer according to the communication instructions. The SCU_router sub-module itself is composed of two routers, as presented in the fig. 3.

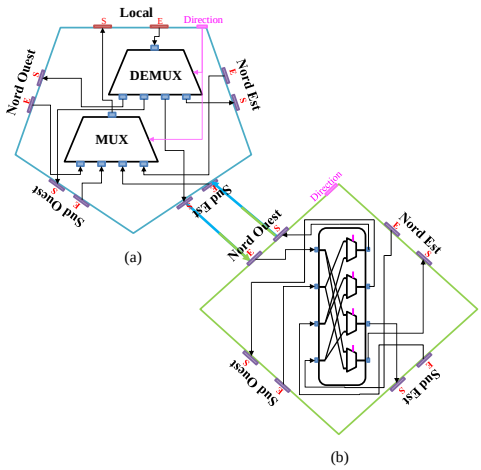

Figure 3: Architecture of the SCU-Router
(a) R-SCUXnet - (b) R-Xnet

All the communications take place in the same direction, so there is no messages congestion in the same data transfer port. This feature allows the simple design of the routing element:

  - **R-SCUXnet** manages directions and distances of communications. It is composed of a couple of 4:1 mux/demux, allowing synchronous data transfer.
  - **R-Xnet** allows simultaneous connection of any pair of non-occupied ports according to the given direction.

Table 3: X-net directions

Direction		Code	R-SCUXnet		R-Xnet	
↖	North West	0	0	↖	0	↖
↑	North	1	0	↖	1	↗
↗	North East	2	1	↗	1	↗
→	East	3	1	↗	2	↘
↘	South East	4	2	↘	2	↘
↓	South	5	2	↘	3	↙
↙	South West	6	3	↙	3	↙
←	West	7	3	↙	0	↖

Each SCU-Router can take 4 different directions, allowing the data transfer in 8 directions, as detailed in table 3.

When the COM-Control decodes the communication instructions, it orders the COM-Router to open the selected ports of the couple (R-Xnet,R-SCUXnet) to achieve the data transfer in the specified direction. A COM-Control handles the communication requests and transfers data from the local SCU to the neighbour SCU in a given direction. Once the communication is established, data will be stored in R_COM register to be used by the requester CE.

- OR_Tree module
  The barrier synchronization is a high latency operation in massively parallel systems. Several systems have implemented either dedicated barrier networks [12] or provided hardware support within existing data networks [13]. The OR-Tree is a mechanism of global OR, checking the state of the system parallel processing. It is composed of a tree of ”OR” gates, which compares the end execution signals of all the CEs in pairs. It is a barrier synchronization that allows the controllers to know if all activated CEs finished the computation. The G-MPSoC supports a hierarchical OR_Tree structure. The first level is in the SCU component to test the end execution in cluster of CEs and the second one is in the SCUs grid to test the end execution in all nodes of the system.

The modular structure of the SCU and the independent execution of the control functions allow the autonomous parallel computation while performing the parallel communication.

3.1.3 X-net: neighbourhood interconnection network

To maintain efficient execution in embedded systems and high performance through a selective broadcast, we propose an on-chip regular neighbourhood interconnection network inspired from the network used in MP-1 and MP-2 MasPar [22] machines: called X-net Network-on-Chip. The X-net network uses various configurations according to the data-parallel algorithms needs. Therefore, we define different bus sizes (1 bit, 4 bits or 16 bits) and different network topologies 1D (linear and ring) and 2D (mesh and torus). To change from one configuration to another, the designer has to use specific parameters (topology and bus size values) to establish the appropriate connections. When the network parameters are selected, the X-net network is generated with the chosen configuration. To achieve the re-usability and reconfigurability, the X-net network directly connects each CE with its 8 nearest neighbours in bidirectional ports through the SCU component. In some cases, the extremity connections in X-net network are wrapped around to form a torus topology, which is used to facilitate the matrix computation algorithms. All SCUs have the same direction controls. In fact, each SCU can simultaneously send a data to the northern neighbour and can receive another data from its southern neighbour. The X-Net uses a bit-state signal to identify nodes that participate in communication. Inactive SCUs can be used as pipeline stages to achieve distant communication. This data transfer through networks occurs without conflicts and is achieved by COM_S and COM_R instructions that allow all the SCUs to communicate with their neighbours in a given direction at a given distance.

3.1.4 Cluster of CEs

The cluster of CEs is the third execution level in G-MPSoC. It allows the parallel execution through either homogeneous CEs, using the same Software SW-CE, or heterogeneous CEs with different Software SW-CE and Hardware HW-CE.

- SW-CE: Processor Element (PE)
  Each PE executes its own instructions block independently from the others PEs. This autonomous execution requires the integration of a local instruction memory and a local Program Counter (PC) in each SW-CE. When the SCU orders the parallel computation, all active SW-CEs start the execution of their local instructions blocks at the same address on different data.

- HW-CE: IP Accelerator
  Some specific functions performed by the processor can be assigned to dedicated hardware modules, called IP accelerators. It allows executing a specific function more efficiently than with any processor. Regardless of the function to achieve, HW-CE is only sensitive to the trigger signal sent by the SCU controller. Once received, the HW-CE begins the execution, independently of the others CEs in the cluster.

The external interface of the CE is the same whatever the nature of the component (HW-CE or SW-CE) to allow the rapid integration of the CE component in the G-MPSoC architecture. Each CE in the cluster is connected to its own data register R_CE_i located in SCU to store the intermediate results, needed for the communication process, and the final result that will be transmitted to the MCU. Each cluster of CEs is controlled by its SCU and only performs when receives the trigger signal.

3.1.5 Memories

- Sequential memory: MCU-memory
  The MCU is connected to the program memory. It includes sequential instructions to be executed by MCU and parallel control instructions to be broadcast to SCUs. The data are stored in the stack of MCU processor.

- Parallel distributed memories: M_i memory
  Each SW-CE is connected to its own local M_i memory. It is divided into an instructions memory, including parallel instructions blocks, and a data memory, including parallel data. Depending on the SW-CE nature, the designer chooses to include or not the data memory. For example, in the case where SW-CE is a stack processor, the use of data memory is not necessary.

The G-MPSoC architecture is designed as parametric and reconfigurable system, which is able to target various applications through customized architecture. Indeed, the number of the nodes (SCUs, CEs and memory size) is parametric in G-MPSoC, allowing the easiest scalability of the system. All these nodes are connected via a reconfigurable interconnection network, which can have one topology among 4 (2D (mesh, torus) and 1D (linear, ring)), 1 direction among 8 (N, S, E, O, NE, NO, SE and SO) and 1 distance ranging between 1 and 16. To facilitate the modification and the reuse of the components that constitute the architecture, a generic implementation is defined, by which several G-MPSoC configurations can be built. To configure a G-MPSoC system from a top level design, the designer defines the entities of the modules that will be used in the system,

specifies their parameters and interconnects them together. Indeed, G-MPSoC instance can range from a simple configuration composed of a MCU connected to a parametric grid of SCUs, where each SCU is connected to a single CE, to a complex configuration where SCUs are interconnected via a reconfigurable neighbourhood network and each SCU is connected to a cluster of heterogeneous CEs. Therefore, from a generic architecture a tailored solution can be built, according to the application requirements, in terms of resources: computation, memorization and communication.

3.2 Execution model of G-MPSoC

3.2.1 Synchronous Communication

- One-to-all communication: broadcast with mask
  The broadcast with mask [11] is a technique of dividing a network into two or more subnetworks, where the execution of different instructions is autonomous. This mechanism starts by activating the SCUs involved in the execution according to the mask sent by the MCUs. Each SCU in the grid has a unique number composed of the reference couple (X,Y). This number represents its position relative to X line and Y column. The mask codes set the activity flag of each SCU. All active SCUs execute control instructions, while the others remain in an idle state.

- Collective regular communication
  The communication in G-MPSoC is defined by the temporal and spatial regularity of its data transfer. It is regular, by which all the nodes have the same degree of connection with their neighbours and synchronously communicate in the same direction and distance. This regular communication is managed by Send and Receive instructions. It is synchronously performed from a single control flow and locally controlled by the slave controllers. Such communication is simple to design without the overhead of synchronization mechanism.

3.2.2 Asynchronous Computation

The master-slave control mechanism [14] is based on two control levels: the first one (MCU) executes sequential instructions and sends parallel control instructions to the second control level: SCUs. The second level controls the parallel communication in the interconnection network and the parallel execution in the cluster of CEs. Each CE executes its instructions stream asynchronously from the others, while the SCUs manage the synchronous data transfer in the network. The master-slave control mechanism provides a flexible parallel execution with the use of multiple control flows globally synchronized. This flexibility increases the system scalability.

4 G-MPSoC implementation

The choice of the G-MPSoC implementation on FPGA is justified by the flexibility and the reconfigurability of this device. It allows the implementation of generic architecture that is effectively tailored to the application requirements. The number of nodes (SCU controllers + CEs calculators) and the local memories size are parametric. Indeed, a targeted application can need many SCUs with many heterogeneous CEs using short memories, or a small amount of SCUs and CEs with large memories. This architecture is implemented with VHDL language and targets the Xilinx Virtex6 ML605 FPGA [15]. To evaluate the G-MPSoC performance, we use the ISim simulator and the ISE synthesis Xilinx tools.

The G-MPSoC configuration includes a single MCU and a grid of nodes. Each node is a hierarchical unit that contains a SCU and its cluster of CEs. Each CE is another hierarchical unit that contains an IP-accelerator or a PE connected to its local memory. Another intermediate hierarchical level, connecting several SCUs, has been defined to facilitate the routing process. All processors, used in G-MPSoC architecture, have a pre-existent FPGA implementation (FC16 [10], HoMade [16]...). We add some signals to be adapted to the CE generic interface and some instructions like: the wait_GO instruction, to wait for the trigger signal sent from the SCU, and the end_CE instruction, to inform the SCU of the end of the parallel computation. MCU is a modified FC16 processor, where mask, broadcast and wait_end instructions are added. Its interface with the SCUs grid is modified to support 32 bits buses and 3 bits mask/broadcast code bus. The IP-accelerator can be a predefined Xilinx IP or an implemented IP using the RTL description. The memories modules are implemented using the existing FPGA blocks memories.

The connections between the components used in this design require several signals that consume a large area. To increase the system scalability, it is necessary to optimize the use of these signals without affecting the processing. We have proposed to break all final result connections from SCUs to MCU and we have only kept the closest signals to MCUs (i.e. the SCUs in the first column). For the others, shift operations are performed to transfer the final result to the SCUs in the first column. This method allows the decrease of the system area occupancy approximately

about 6.26%.

The Synthesis results predict the maximum frequency of a given configuration, ranged around 88.212MHz This frequency is relative to the used processors frequency (FC16 151.404MHz, HoMade 94.117MHz) and the longest critical path in the design. A good place&route of the designed components is necessary to reduce the length of this critical path and to accelerate the signal propagation.

Table 4 gives the resources occupancy for the G-MPSoC components on the targeted FPGA. Depending on the used CE size, G-MPSoC configuration can include 100 nodes with FC16 processor, 49 nodes with HoMade processor or more than 256 nodes with muladd accelerator. This number of component can be increased with the use of another FPGA generation with more provided hardware resources to reach the defined architecture with 256 nodes and 16 CEs per cluster (i.e. 4096 CEs). The table 4 shows the power consumption of these modules. For all of them, the power characteristics are reported as around 400mW. This low power consumption makes the choice of the integrated CE only depending on size and speed. For simple operation, it is better to integrate specialized accelerator IPs than processors, in order to raise the architecture size and to accelerate the parallel execution. Table 4 also shows that the control structure in G-MPSoC system does not clutter the occupied area. In particular, the SCU module contains four sub-modules that not require too much logic elements, as shown in table 5. Thereby, the total area-cost of G-MPSoC system does not burst with the use of master-slave control structure. In fact, based on the work in [14], the experimental results show that for 100 nodes with clusters of a single CE (FC16 processor) connected to 4KB local memory, the grid of SCUs occupy about 38% of the total consumed on-chip logic area. For an array of 16 SCUs, it is around 16%. In fact, if the number of G-MPSoC nodes increases, the area occupancy linearly increases, but the incremental cost of adding SCU functionality to G-MPSoC control system quickly becomes small.

Table 5: SCU sub-modules: synthesis result on FPGA Virtex6 ML605

Sub-module	LUTs	Registers
Activity controller	4	2
Parallel execution controller	352	80
Communication controller	101	50
End execution controller (ORTree)	3	0

5 Experimental results

The generic feature of G-MPSoC is ensured not only by the modular and parametric structure of its different components, but also by its hierarchical-distributed control, the diversity and the heterogeneity of its processing units, and the parametricity of its interconnection network. In this section and through several benchmarks, we validate the generic feature of the architecture and show the effectiveness of their implementation in terms of flexibility, scalability and high performance (execution time and bandwidth). To test these benchmarks, we designed the G-MPSoC architecture with VHDL language and targeted the Xilinx Virtex6-ML605 FPGA as prototyping platform.

5.1 Benchmark 1: Red-Black checkerboard

Master-slave control in G-MPSoC architecture is based on a hierarchical-distributed structure, which allows having several parallel processing areas selected by the sub-netting mechanism. Thereby, the MCU can manage these areas and can switch from one to another using different activity masks, during the execution of a parallel program. This flexibility to switch between the processing areas allows the simultaneous execution of both conditional structure blocks *(if...then...else...)*. This feature facilitates the activity process and avoids having idle processing nodes when executing the conditional structure in massively parallel architecture.

To highlight this feature and its impact on the proposed architecture performance, we tested the Red-Black checkerboard application with the broadcast-with-mask [11] and traditional one-to-all [19] methods. This application is used to solve partial differential equations (Laplace equation with Dirichlet boundary conditions, Poisson-Boltzmann Equation, etc), using massively parallel systems. It is based on the division of parallel processing nodes into red and black areas, where all the same colour area are performing the same instructions block. The code below is composed of a set of the added mask/broadcast instructions and a "lit" FC16 instruction [10] to map the red-black mask into the processing grid, and then broadcast the parallel instruction to trigger the execution of the first instructions block. Using the inverted mask, the second area can be activated and the execution of the second instructions block can be triggered.

The code of the broadcast-with-mask, detailed in listing 1, is implemented with only 6 instructions to subnet the (16×16) grid into red-black checkerboard as shown in fig. 4. Therefore, the nodes with the same colour have the same control flux and execute the same instructions blocks, independently of the others

Table 4: G-MPSoC components: synthesis result on FPGA Virtex 6 ML605

		LUTs	Slice registers	DSP48E1s	FPGA occupancy	Fmax(Mhz)	Power(W)
CE	FC16	1132	206	-	<1%	151.404	0.462
	HoMade	3560	493	1	2%	94.117	0.509
	muladd accelerator	26	19	1	<1%	624.220	0.436
Control module	MCU	1139	201	-	<1%	153.520	0.483
	SCU	420	132	-	<1%	256.937	0.536

nodes in the neighbour sub-network.

Listing 1: Broadcast-with-mask code for Red-Black checkerboard

```
lit 0xAAAAAAAA  // find mask A (2 cycles)
selbf           // send mask A (1 cycle)
lit 0x55555555  // find mask B (2 cycles)
selbfor         // send mask B (1 cycle)
lit 0x06000010  // find parallel control
    instruction (2 cycles)
brdbf           // send parallel control
    instruction (1 cycle)
```

As shown in listing 1, the red-black mask can be rapidly mapped into a grid of (16 × 16) nodes in 6 clock cycles, unlike the one-to-all method [19], that needs several clock cycles to map this mask into the processing nodes. With the one-to-all method, there must be a relationship between identity and the activity bit to activate the nodes with peer identities and disable the ones with odd identities, or to enable nodes with identity lower than a specific value and disable the others. So that to map the Red-Black mask into a grid of (16 × 16) nodes, 16 operations are required: 8 for odds and 8 for peers, executed in 20 clock cycles. Thus, we notice that the broadcast-with-mask method allows the rapid grid sub-netting on several processing areas, twice shorter than with the traditional one-to-all method.

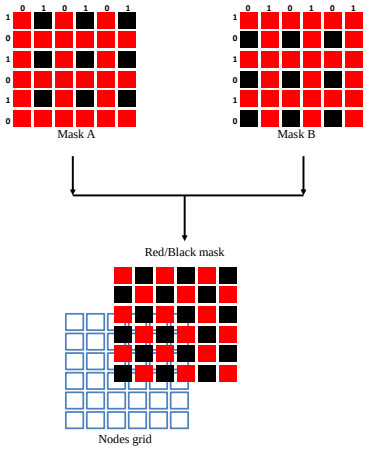

Figure 4: Red-Black mask

The parallel instructions broadcast using these two different activity control methods requires different implementations of the control structure in G-MPSoC. The synthesis result given by ISE tool [17] and presented in table 6 shows the large bandwidth (∼12% higher than the one-to-all) provided by the G-MPSoC using broadcast-with-mask despite the additional consumed logic elements. This can be explained by the fact that the proposed control structure for parallel activity and parallel broadcast management is based on local controllers (SCUs), which optimize the data flow transfer.

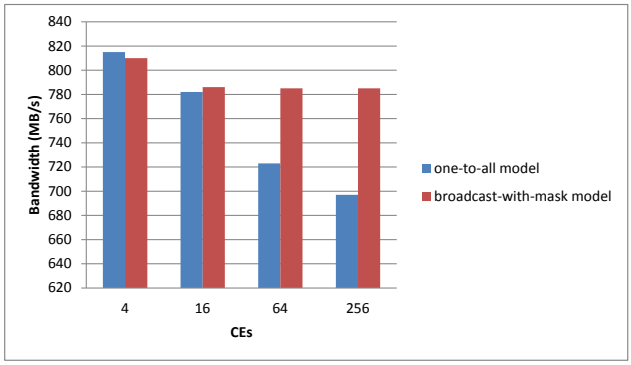

Figure 5: Influence of broadcast models on bandwidth

Fig. 5 shows that an increasing number of nodes integrated in the G-MPSoC architecture leads to a higher gap between the bandwidth provided by the broadcast-with-mask method and one-to-all method. This result shows the importance of broadcast-with-mask technique in the massively parallel systems scalability, without causing bottlenecks. However, the traditional one-to-all method is recommended with small systems because of its efficiency in terms of area cost and bandwidth. Thereby, the designer has to make the right choice between the system size and the broadcast technique to ensure the high performance needed for the parallel execution.

5.2 Benchmark 2: Matrix multiplication

The processing units (CEs) in the G-MPSoC architecture can have several forms. They can be homogenous with the same type of processors or heterogeneous with clusters of different processors or clusters of dif-

Table 6: Synthesis results on FPGA Virtex6 of G-MPSoC with different activity control methods

	FPGA occupancy			Performance		
	LUTs	Slice registers	memories	Fmax(Mhz)	Bandwidth(MB/s)	Power(W)
Broadcast with mask	257747 54%	41090 4%	10256 7%	205.999	786	0.421
Broadcast one-to-all	242768 51%	41056 4%	10256 7%	182.846	697	0.447

ferent processors and accelerators. Each one performs a specific function or blocks of parallel instructions.

To highlight the genericity and the flexibility of the CEs, we have tested the matrix multiplication example with two different G-MPSoC configurations with a grid of (8×8) nodes, where each node is composed of:

- Conf.1: SCU + software CE (FC16 processor).
- Conf.2: SCU + software CE (FC16 processor) + hardware CE (muladd accelerator to perform the multiplication and the addition operations).

The matrix multiplication is one of the basic computational kernels in many data parallel applications. It is presented by the equation (1):

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad avec \quad 1 \leq i, j \leq n \qquad (1)$$

To perform (8×8) matrix multiplication into system with (8×8) nodes, the execution needs 16 multiplications, 16 additions and 30 communications. The FC16 processor performs the multiplication in 19 clock cycles [10], whereas muladd accelerator performs the multiplication in one cycle. That is why the configuration based on clusters of (FC16 + muladd) is more efficient than the architecture based on only SW-CE (FC16). We notice that the conf.2 is the most suitable for matrix multiplication.

The generic feature of G-MPSoC architectural model allows changing the system structure from a homogeneous configuration to a heterogeneous configuration according to the performance provided by the CEs and the algorithm requirements. The use of parametric modules significantly facilitates the generation of processing nodes with rapid modification of system configuration. Thereby, the G-MPSoC system is flexible, scalable and quickly adapts to applications changes.

5.3 Benchmark 3: Parallel Summing

To validate the flexible communication in G-MPSoC architecture, we chose the summing application [18] using a grid of 4×4 nodes. The aim of this application is to test the synchronous communication in G-MPSoC system with different directions and distances and using the Send/Receive instructions. This application is performed via the X-net network, using the neighbourhood/distant communications and the broadcast-with-mask method to facilitate the network sub-netting, as shown in fig. 6.

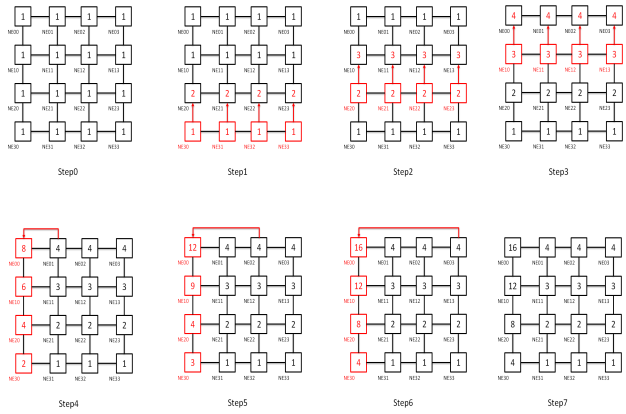

Figure 6: Summing application steps

Performing the Summing application, in traditional SIMD system, needs several clock cycles, especially for node activity step and communication step. In addition, doing distant communication in [18] requires external link. Therefore, the use of the X-net network and the broadcast-with-mask mechanism improves the communication performance in G-MPSoC architecture, where X-net communication costs *d* (distance) cycles: the delay of data transfer between source and destination (1 cycle for neighbour communication).

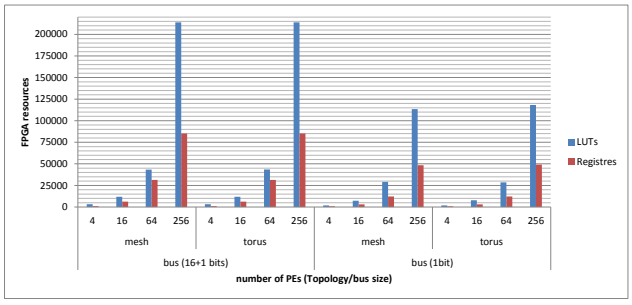

Figure 7: FPGA occupancy with different size of G-MPSoC configurations

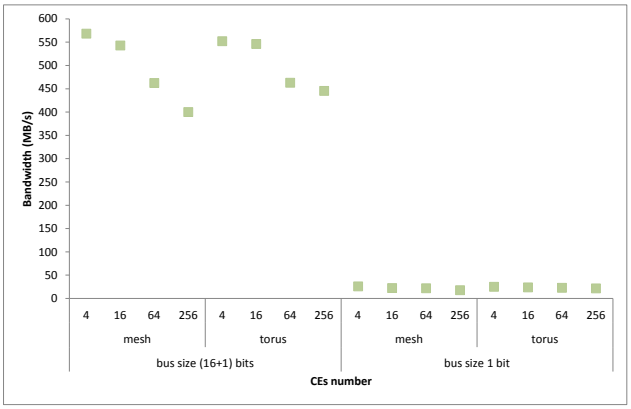

Figure 8: Influence of CEs number and bus sizes on bandwidth

Fig. 7 and fig. 8 present synthesis and bandwidth results on different system topologies and bus sizes. We note a compromise between area and bandwidth. Indeed, the configuration integrating the X-net interconnect with (16+1) bits buses gives efficient data transfer with large bandwidth, but it occupies a large chip area (2 times higher), as shown in fig. 7. In addition, if the number of the nodes is multiplied by a factor of 16, the bandwidth decreases by factor of 2 and the FPGA area increases by a factor of 4, which is an acceptable rate.

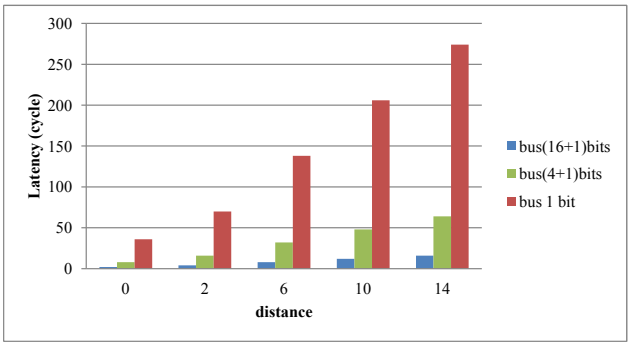

Figure 9: Influence of buses size on communication delay

We have also tested the communication delay with the use of the different bus sizes. Fig. 9 shows that the communication time is 17 times higher with 1-bit bus than with 17-bits bus. Despite this tedious communication, 1-bit data transfer allows the use of relatively simple buses with low hardware cost, which increase system scalability. These previous experimental results show the efficiency of the integration of reconfigurable interconnection network in G-MPSoC. Depending on the application needs, the designer can select the network parameters, which guarantee the less communication delay.

5.4 Benchmark 4: FIR filter

The digital Finite Impulse Response (FIR) [21] filters are widely used in digital signal processing to reduce certain undesired aspects. A FIR structure is described by the differential equation (2):

$$y(n) = \sum_{k=0}^{N-1} h(k) \times x(n-k) \qquad (2)$$

It is a linear equation, where X represents the input signal and Y presents the output signal. The order of the filter is given by the parameter N, and $H(k)$ represents the filter coefficients. The number of inputs and outputs data is equal to n. To emphasize the advantage of communication networks in the execution of data-parallel programs, we have implemented the FIR filter with two methods:

1st method: 2D configuration

This method is inspired from the work presented in [20]. It is an implementation of FIR filter as defined in its equation (2). The system is composed of a MCU, a grid of (4×4) nodes (SCU + PE) and an X-net network in torus topology. We assume that the FIR system takes 16 $H(k)$ parameters and 64 $X(n)$ inputs. The algorithm is described as the following steps:

1. Data initialization: each $H(k)$ is stored in *PE(i,j)* local data memory according to the following function: $k = 4 \times i + j$ and all $X(n)$ values are stored in each PE memory.

2. Multiplication of all the inputs with *H(k)* in each *PE(i,j)*.

3. Communication: as shown in fig. 10, all PEs do West communication to send first multiplication element to their neighbour; then only PEs in the last column do North-communication.

4. Addition of this new value with the second local multiplication value in each *PE(i,j)*.

5. Repeat 4) until obtain all the results which will be stored in *PE(0,0)* memory.

To perform this FIR filter algorithm, (64×2) communications are required. The multiplication and the addition operations are performed in parallel.

2nd method: 1D configuration

This method is inspired from the work presented in [23]. In this implementation, the system is composed of a MCU, 16 slaves and X-net network in linear topology. We also assume that the FIR system takes 16 $H(k)$ parameters and 64 $X(n)$ inputs. All $H(k)$ are stored in MCU and sent to *PE(i)* when

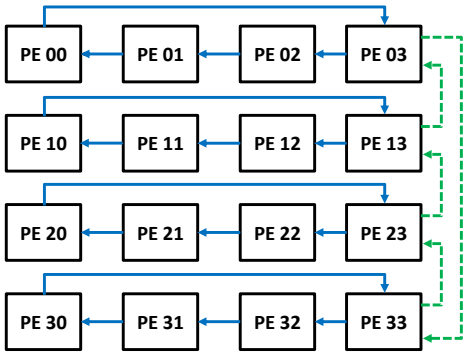

Figure 10: FIR filter implementation in 2D-configuration

Table 7: 16-order FIR with inputs implementation

the inputs (n)	Com execution time (cycle) on SIMD mode			
	G-MPSoC Method 1	G-MPSoC Method 2	ESCA	reconf-SIMD on-chip
8	126	72	255	332
16	270	144	351	744
64	1134	576	1158	-

they are needed in the algorithm. The input $X(n)$ are shifted inter-slaves using West-communication, as presented in fig. 11. The n outputs are calculated using multiplication and addition instructions, in the alternative way. To do this algorithm, only 63 communications are required but the algorithm is not totally parallel.

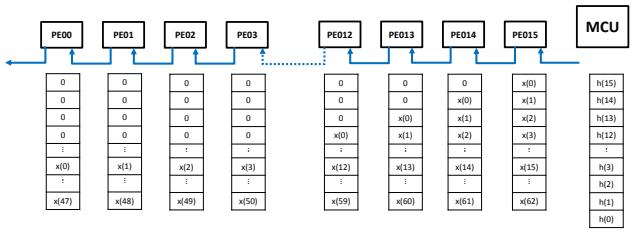

Figure 11: FIR filter implementation in 1D-configuration

Experimental results

The experimental results in table 7 show the time needed for data transfer in FIR filter application. As expected, the G-MPSoC architecture allows more rapid processing than both reconfigurable SIMD architecture on-Chip [23] and ESCA [20] architectures. We deduce that G-MPSoC architecture based on linear interconnection topology is the most effective for the FIR application.

Depending on the application needs, the designer can select the most appropriate network configuration to his system. The different topologies that can support the X-net offer diverse choice of 1D or 2D configurations as well as at the size of the interconnect bus, to ensure data transfer rapidly and with low cost.

6 Conclusion

This paper presents a new generation of massively parallel System-on-Chip based on generic structure, called G-MPSoC platform. It is a configurable system, composed of clusters of hardware and/or software CEs, locally controlled by a grid of SCUs and globally orchestrated by the MCU. All the CEs can communicate between each other via the SCUs components that are connected through regular X-net interconnection network. G-MPSoC architecture is entirely described in VHDL Language, in order to allow rapid prototyping and testing, using the synthesis and simulation tools. The execution model of G-MPSoC is detailed to highlight the advantages of the synchronous communication, based on broadcast with mask structure and regular communication network, and asynchronous computation, based on master-slave control structure. An FPGA Hardware implementation of G-MPSoC platform is also presented in this paper and validated through several parallel applications. Different configurations are tested going from a simple homogeneous structure to a complex heterogeneous structure. All configurations tested different X-net network topologies, different data buses sizes and different memories sizes. In this work, we define a generic massively parallel system on chip that is able to be quickly adapted to the applications requirements.

The next work is to define a new weakly coupled massively parallel execution model for G-MPSoC platform, based on Synchronous Communication and Asynchronous Computation. This model is classified between the synchronous centralized SIMD model and the asynchronous decentralized MIMD model. It takes advantages of these two models to allow more performance needed for the execution of nowadays intensive processing applications.

References:

[1] D. Melpignano, L. Benini, , E. Flamand, et al. *Platform 2012, a many-core computing accelerator for embedded SoCs: performance evaluation of visual analytics applications*. Proc. Int. Conf. Design Automation Conference, New York, USA, 2012, pp. 1137–1142.

[2] *SIMD < SIMT < SMT: parallelism in NVIDIA GPUs*. http://yosefk.com/blog/simd-simt-smt-parallelism-in-nvidia-gpus.html

[3] *PU-based Image Analysis on Mobile Devices*. http://arxiv.org/pdf/1112.3110v1.pdf.

[4] D.B. Thomas, L. Howes and W. Luk. *A Comparison of CPUs, GPUs, FPGAs, and Massively Parallel Processor Arrays for Random Number Generation*. Proc. Int. Conf. Field Programmable Gate Arrays, New York, USA, 2009, pp. 63–72.

[5] F. Hannig, V. Lari, S. Boppu, et al. *Invasive Tightly-Coupled Processor Arrays: A Domain-Specific Architecture/Compiler Co-Design Approach*, ACM transactions on Embedded Systems, 13, 2014, pp. 133:1-133:29.

[6] F. Conti, C. Pilkington, A. Marongiu, et al. *He-P2012: Architectural Heterogeneity Exploration on a Scalable Many-Core Platform.* Proc. Int. Conf. Application-specific Systems, Architectures and Processors, Zurich, Swiss, 2014, pp. 114-120.

[7] *Many-core Kalray MPPA*, http://www.kalray.eu

[8] *The HyperCore Processor*, http://www.plurality.com/hypercore.html

[9] *Next Generation CUDA Compute Architecture: Fermi WhitePaper*, http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/NVIDIA-Fermi-Compute-Architecture-Whitepaper-en.pdf

[10] R.E. Haskell and D.M. Hanna, *A VHDL Forth Core for FPGAs*, Journal of Microprocessors and Microsystems, 29, 2009, pp. 115-125.

[11] H. Krichene, M. Baklouti, Ph. Marquet, et al. *Broadcast with mask on Massively Parallel Processing on a Chip*. Proc. Int. Conf. High Performance Computing and Simulation, Madrid, Spain, 2012, pp. 275–280.

[12] C.E. Leiserson, Z.S. Abuhamdeh, D.C. Douglas, et al. *The network architecture of the connection machine CM-5*, Journal of Parallel and Distributed Computing, 33, 1996, pp. 145-158.

[13] S.L. Scott. *Synchronization and communication in the T3E multiprocessor*. Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems, New York, USA, 1996, pp. 26–36.

[14] H. Krichene, M. Baklouti, Ph. Marquet, et al. *Master-Slave Control structure for massively parallel System-on-Chip*. Proc. Int. Conf. Euromicro Conference on Digital System Design, Santander, Spain, 2013, pp. 917–924.

[15] *ML605 Hardware User Guide - UG534 (v1.8)*, http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf.

[16] *HoMade processor*, https://sites.google.com/site/homadeguide/home.

[17] *Xilinx website*, https://www.xilinx.com.

[18] M. Leclercq and P.Y. Aquilanti, *X-Net network for MPPSoC*, Master thesis, University of Lille 1, 2006.

[19] M. Baklouti, *A rapid design method of a massively parallel System on Chip: from modeling to FPGA implementation*. PhD thesis, University of Lille 1 & University of Sfax, 2010.

[20] P. Chen, K. Dai, D. Wu, et al. *Parallel Algorithms for FIR Computation Mapped to ESCA Architecture*. Proc. Int. Conf. Information Engineering, Beidaihe, China, 2010, pp. 123–126.

[21] S.M. Kuo and W.S. Gan, *Digital Signal Processors Architectures, implementation and application*. Prentice Hall, 2005.

[22] J.R. Nickolls, *The design of the MasPar MP-1: a cost effective massively parallel computer*, IEEE Computer Society International Conference, Compcon Spring, San Francisco, USA, 1990, pp. 25–28.

[23] J. Andersson, M. Mohlin and A. Nilsson, *A reconfigurable SIMD architecture on-chip*. Master Thesis in Computer System Engineering - Technical Report, School of Information Science, Computer and Electrical Engineering - Halmstad University, 2006.

[24] Sh. Raghav, A. Marongiu and D. Atienza, *GPU Acceleration for Simulating Massively Parallel Many-Core Platforms*. Journal of Parallel and Distributed Systems, 26, 2015, pp. 1336–1349.

[25] D. Walsh and P. Dudek , *An Event-Driven Massively Parallel Fine-Grained Processor Array*. Proc. Int. Conf. Circuits and Systems (ISCAS), Lisbon, Portugal, 2015, pp. 1346–13496.